Telnet is an application layer protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection. User data is interspersed in-band with Telnet control information in an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP). Telnet was developed in 1969 beginning with RFC 15, extended in RFC 854, and standardized as Internet Engineering Task Force (IETF) Internet Standard STD 8, one of the first Internet standards. [8]

Following example depict how to **telnet** command line to send a simple HTTP GET Request to `http://httpbin.org/status/418` webpage.

```
> telnet httpbin.org 80

GET /status/418 HTTP/1.0
Host: httpbin.org

HTTP/1.1 418 I'M A TEAPOT
Server: nginx
Date: Sat, 29 Jul 2017 21:58:24 GMT
Content-Length: 135
Connection: close
Access-Control-Allow-Origin: *
x-more-info: http://tools.ietf.org/html/rfc2324
Access-Control-Allow-Credentials: true


    -=[ teapot ]=-

       _...._
     .'  _ _ `.
    | ."` ^ `". _,
    \_;`"---"`|//
      |       ;/
      \_     _/
        `"""`
Connection closed by foreign host.
```

To reproduce the example, your should follow the following steps:

1. Launch your terminal/command console. Most operating systems have the **telnet** command line installed. If this is not the case on your version, you need to install it.
2. In the terminal type `telnet httpbin.org 80`, where httpbin.org is the HTTP server and '80' is the standard HTTP TCP port.
3. Once **telnet** connected, type `GET /status/418 HTTP/1.0`, then press Enter in the keyboard.
4. Afterward, type `Host: httpbin.org` then two times Enter to send the request to **httpbin.org** HTTP Server.

You should receive the response depicted in the output sample. Similarly, you can explore HTTP protocol operations without the need to program.

HTTP Client Library Implementation

After getting a deeper understanding of the flow of HTTP GET and POST operations, you are requested to implement your HTTP client using TCP Sockets. The programming library implements only a small subset of HTTP specifications. In other words, we expect that your HTTP library supports the following features:

1. GET operation
2. POST operation
3. Query parameters
4. Request headers
5. Body of the request

You can refer to HTTP protocol reference for more information about these features. You can start your implementation by leveraging the provided samples for HTTP-ECHO and HTTP-TIME protocols at the example directory.

---

cURL-like Command Line Implementation

At this stage, your ready to build a simple HTTP client using your library. In this task, you are requested to implement cURL command line with basic functionalities.

For this purpose, you need to experiment with cURL command line in your preferred terminal in order to know its basic options. You should notice that our focus is on the options related to HTTP protocol since cURL support many network protocols as described earlier.

The implemented client should be named **httpc** (the name of the produced executable). The following presents the options of your final command line.

```
httpc (get|post) [-v] (-h "k:v")* [-d inline-data] [-f file] URL
```

In the following, we describe the purpose of the expected **httpc** command options:

1. Option **-v** enables a verbose output from the command-line. Verbosity could be useful for testing and debugging stages where you need more information to do so. You define the format of the output. However, You are expected to print all the status, and its headers, then the contents of the response.

2. **URL** determines the targeted HTTP server. It could contain parameters of the HTTP operation. For example, the URL 'https://www.google.ca/?q=hello+world' includes the parameter **q** with "hello world" value.
3. To pass the headers value to your HTTP operation, you could use **-h** option. The latter means setting the header of the request in the format "*key*: *value*." Notice that; you can have multiple headers by having the **-h** option before each header parameter.
4. **-d** gives the user the possibility to associate the body of the HTTP Request with the inline data, meaning a set of characters for standard input.
5. Similarly to **-d**, **-f** associate the body of the HTTP Request with the data from a given file.
6. **get**/**post** options are used to execute GET/POST requests respectively. **post** should have either **-d** or **-f** but not both. However, **get** option should not used with the options **-d** or **-f**.

Detailed Usage

General

```
httpc help
```

```
httpc is a curl-like application but supports HTTP protocol only.
Usage:
    httpc command [arguments]
The commands are:
    get     executes a HTTP GET request and prints the response.
    post    executes a HTTP POST request and prints the response.
    help    prints this screen.

Use "httpc help [command]" for more information about a command.
```

Get Usage

```
httpc help get
```

```
usage: httpc get [-v] [-h key:value] URL

Get executes a HTTP GET request for a given URL.

   -v             Prints the detail of the response such as protocol, status,
and headers.
   -h key:value   Associates headers to HTTP Request with the format
'key:value'.
```

Post Usage

```
httpc help post
```

```
usage: httpc post [-v] [-h key:value] [-d inline-data] [-f file] URL

Post executes a HTTP POST request for a given URL with inline data or from
file.

   -v             Prints the detail of the response such as protocol, status,
and headers.
   -h key:value   Associates headers to HTTP Request with the format
'key:value'.
   -d string      Associates an inline data to the body HTTP POST request.
   -f file        Associates the content of a file to the body HTTP POST
request.

Either [-d] or [-f] can be used but not both.
```

Examples

Get with query parameters

```
httpc get 'http://httpbin.org/get?course=networking&assignment=1'
```

Output:

```
{
  "args": {
    "assignment": "1",
    "course": "networking"
  },
  "headers": {
    "Host": "httpbin.org",
    "User-Agent": "Concordia-HTTP/1.0"
  },
  "url": "http://httpbin.org/get?course=networking&assignment=1"
}
```

Get with verbose option

```
httpc get -v 'http://httpbin.org/get?course=networking&assignment=1'
```

Output:

```
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 1 Sep 2017 14:52:12 GMT
Content-Type: application/json
Content-Length: 255
Connection: close
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

```
{
  "args": {
    "assignment": "1",
    "course": "networking"
  },
  "headers": {
    "Host": "httpbin.org",
    "User-Agent": "Concordia-HTTP/1.0"
  },
  "url": "http://httpbin.org/get?course=networking&assignment=1"
}
```


Post with inline data

```
httpc post -h Content-Type:application/json --d '{"Assignment": 1}'
http://httpbin.org/post
```

Output:


```
{
  "args": {},
  "data": "{\"Assignment\": 1}",
  "files": {},
  "form": {},
  "headers": {
    "Content-Length": "17",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "Concordia-HTTP/1.0"
  },
  "json": {
    "Assignment": 1
  },
  "url": "http://httpbin.org/post"
}
```

Optional Tasks

If you have successfully completed the material above, congratulations! The reason behind such congratulations is that you now understand the implementation of an HTTP client and network protocols in general. For the rest of this lab exercise, we have included the following optional tasks. These optional tasks will help you gain a deeper understanding of the material, and if you can do so, we encourage you to complete them as well. Bonus marks will be given for that.

Enhance Your HTTP Client library

In the current HTTP library, you already implemented the necessary HTTP specifications, GET and POST. In this optional task, you need to implement one new specification of HTTP protocol that is related to the client side. For example, you could develop one of the Redirection specifications. The latter allow your HTTP client to follow the first request with another one to new URL if the client receives a redirection code (numbers starts with 3xx). This option is useful when the HTTP client deal with temporally or parental moved URLs. Notice, you are free to choose which HTTP specification to implement in your HTTP library. After selecting the specification, you should consult with the Lab Instructor before starting their implementations.

Update The cURL Command line

Accordingly, you could add the newly implemented HTTP specifications in your HTTP library to the **httpc** command line. To do that, you need to create a new option that allows the user the access the newly implemented specification. In addition, you are requested to add the option **–o filename**, which allow the HTTP client to write the body of the response to the specified file instead of the console. For example, the following will write the response to **hello.txt**:

```
httpc -v 'http://httpbin.org/get?course=networking&assignment=1' -o hello.txt
```